

# *Headed Span Theory in the Finite State Calculus*

Mats Rooth

Universtity of Delaware, March 20, 2014

## OT derivations

/Mawapa/ can be pronounced with any six segments.

A random sample from the set of candidates C0:

7.5 Kb. 43 states, 228 arcs, 72783360 paths.

(.a)[.p][.wwml]

[.f](.w)[.l][w.fw]

(l.w)[.l][.w](.ap)

(l.m)(a.a)[l.m]

[.p][.f][l.l][.m](.f)

(.p)[.f](.w)[.w][.l][.f]

(.a)(.wfl)(a.a)

# Mawapa

Filter away candidates that violate faithfulness for consonants, to produce C7. A random sample: 5.6 Kb. 43 states, 148 arcs, 336960 paths.

[mww.p](.pm)

[.m](a.w)[.mp](.m)

[m.f](.w)(.f)(.pp)

[.mw](w.m)[.p](.w)

(m.a)(w.wp)(.f)

[.m][.p][wap.p]

# Mawapa

Filter away candidates that violate faithfulness for vowels. A random sample from C8:

3.6 Kb. 43 states, 68 arcs, 1560 paths.

[.m][a.w][a.pa]

(.m)[.a][wa.pa]

[maw.a](.p)(.a)

[.ma](.w)(.a)[.pa]

[.m](.a)(w.ap)[.a]

## Mawapa

Underlying [Nasal +] is realized faithfully.  
Square brackets mark a nasal span, and round brackets an oral (non-nasal) span.

A random sample from C9:

3.4 Kb. 40 states, 61 arcs, 571 paths.

[.m][.a](.wa)[.p][.a]

[.ma](.wap)(.a)

[.ma](.w)(.a)(p.a)

[.ma](.w)[.a][.p](.a)

[.m](.a)(.wap)(.a)

Nasal intervals/spans are marked with square brackets. *m* is always in a nasal span.

# Mawapa

Obstruents head an oral span.

A random sample from C10:

3.0 Kb. 37 states, 51 arcs, 168 paths.

[.m](a.w)(a.p)(.a)

[.m](.a)(.w)(a.p)[.a]

[.m](.a)(.w)(.a)(.pa)

[.ma][.w](.a)(.p)[.a]

[.m][a.w](a.p)(.a)

p is always the head (marked with a preceding period) of an oral span (marked with round brackets).

## Mawapa

Minimize number of nasality spans.  
The minimum is two, given the earlier constraints. The four candidates in C11:  
1.7 Kb. 19 states, 21 arcs, 4 paths.

[.mawa](.pa)

[.maw](a.pa)

[.ma](wa.pa)

[.m](awa.pa)

# Mawapa

Oral spans are left-headed.

This results in a unique solution in C12.

1.3 Kb. 13 states, 12 arcs, 1 path.

[.mawa](.pa)

-- . . . . .



# Headed Span Theory

Headed span theory in phonology is an account of the phonological substance that represents an autosegment such as a nasality feature as a labeled interval in a line, rather than as a vertex in a graph. The intervals (or spans) have distinguished head positions.

J. McCarthy (2004), Headed Spans and Autosegmental Spreading.

# Today's Project

Work out a detailed, computationally executable construction of span theory in the finite state calculus. This includes a construction the constraint families of headed span theory as operators. The finite state calculus is an extended mathematical language of regular expressions.

# Idea 1: Labeled Brackets

Use a string encoding of span representations, using labeled brackets.

[.mawa](.pa)

[NMn.**N**+Mn..NM.NM...NM.M] [N-Mv.Mv..NM.M...NM.M]  
[N-Mg.Mg..NM.M...NM.M] [N-Mv.Mv..NM.M...NM.**NM**]  
[N-Mo.**N**-Mo..NM.NM...NM.M] [N-Mv.Mv..NM.M...NM.**NM**]

# Labeled Brackets

## Start and end of [Nasal -]

[N-Mo . **N**-Mo . . NM . NM . . . NM . M] [N-Mv . Mv . . NM . M . . . NM . **N**M]

## Start and end of [Manner obstruent]

[N-Mo . N-**Mo** . . NM . NM . . . NM . **M**] [N-Mv . Mv . . NM . M . . . NM . NM]

## Start and end of [Manner vowel]

[N-Mo . N-Mo . . NM . NM . . . NM . M] [N-Mv . **Mv** . . NM . M . . . NM . **N**M]

## Idea 2: Underlying representation in the same string and encoding

Start and end of underlying [Nasal -]

[N-Mo . N-Mo . . NM . NM . . . NM . M] [N-M<sub>v</sub> . M<sub>v</sub> . . NM . M . . . NM . NM]

Start and end of underlying [Manner obstruent]

[N-Mo . N-Mo . . NM . NM . . . NM . M] [N-M<sub>v</sub> . M<sub>v</sub> . . NM . M . . . NM . NM]

# Encoding of a minimal timing unit

1. Feature spans that start here underlyingly, with values

[N-Mo . N-Mo . . NM . NM . . . NM . M]

2. Feature spans that start here on the surface, with values

[N-Mo . N-Mo . . NM . NM . . . NM . M]

# Encoding: heads

4. Feature spans that are headed here underlyingly

[N-Mo . N-Mo . . **NM** . NM . . . NM . M]

5. Feature spans that are headed here on the surface

[N-Mo . N-Mo . . NM . **NM** . . . NM . M]

# Encoding: ends of spans

## 8. Feature spans that end here underlyingly

[N-Mo . N-Mo . . NM . NM . . . NM . M]

## 9. Feature spans that end here on the surface

[N-Mo . N-Mo . . NM . NM . . . NM . M]



## Idea 3: Finite state calculus

The finite state calculus is a formal language of extended regular expressions. Define the syntax of phonological representations with a sequence of definitions.

### Feature blocks

```
define Block (N)(M);  
define ValBlock (N [Plus | Minus])(M [n | o | f | l | g | v]);
```

## Idea 3: Finite state calculus

### Left semi-segment

```
# Construct a left semi-segment
# Arguments are value blocks (first two) and blocks.
define LeftSemi(ValUnd,ValSur,Exc,Und,Sur)
  "[" ValUnd Dot ValSur Dot Exc Dot Und Dot Sur Dot;
# Set of left semi-segments
define LeftSemiSeg LeftSemi(ValBlock,ValBlock,Block,Block,Block);
```

## Finite state calculus

The definitions are mathematical, but they can be interpreted computationally using a finite state programming language or toolkit. I use Xfst and Thrax.

```
xfst[9]: regex LeftSemiSeg;  
2.4 Kb. 21 states, 47 arcs, 28224 paths.  
xfst[10]: print random-words  
[N+Mn.N+Mf.M.N..  
[N+Mv.N-Mv..NM.N.  
[Mf.Mn.N.M.N.  
[N+Mv..M.M.M.  
[Mg.N+Mo.NM.NM.M.  
[.N+Mn.NM.M.NM.  
[N+Mf.N-.N.M.NM.
```

## Well-formed words

Well-formed words are defined with a sequence of definitions. The first group define semi-segments (half segments) with properties such as starting and F span or heading an F span underlyingly or on the surface.

```
# left semi-segment not specifying F on surface  
define l0(F) LeftSemi(ValBlock,vb0(F),Block,Block,Block);
```

```
# left semi-segment specifying F on surface  
define l1(F) LeftSemi(ValBlock,vb(F),Block,Block,Block);
```

## Well-formed words

# Left semi-segment that does not head an F span on the surface.  
define h0(F) LeftSemi(ValBlock,ValBlock,Block,Block,b0(F));

# Left semi-segment that heads an F span on the surface.  
define h1(F) LeftSemi(ValBlock,ValBlock,Block,Block,b1(F));

## Well-formed words

```
# right semi-segment not specifying F on surface  
define r0(F)  RightSemi(Block,Block,b0(F));
```

```
# right semi-segment specifying F on surface  
define r1(F)  RightSemi(Block,Block,b1(F));
```

## Well-formed underlying F-span

```
define S(F) [L1(F) [R0(F) L0(F)]* R1(F)] & [[H0(F) RightSemiSeg]*  
H1(F) RightSemiSeg [H0(F) RightSemiSeg]*];
```

## Well-formed word

```
# An F-well-formed word is a sequence of F spans.  
# Well-formed for F on surface.  
define w(F) s(F)+;  
# Well-formed for F underlyingly.  
define W(F) S(F)+;
```



## Well-formed word

A well-formed word is well-formed in each feature, underlyingly and on the surface.

```
# Well-formed words in the nasal problem.  
define Wfw [w(N) & W(N) & w(M) & W(M)];
```

## Well-formed two-segment words

There are a lot, because underlying  
and surface words are not correlated except (in this model) in length.

```
xfst[10]: regex [Seg Seg] & Wfw;
```

```
103.2 Kb. 1494 states, 2282 arcs, 2820096 paths.
```

```
xfst[11]:
```

```
xfst[11]: print random-words
```

```
[N+Mg.N+Mg.NM.M.M..M.M.M][Mf.Mv..NM.NM..N.NM.NM]
```

```
[N+Mn.N-Mg..NM.N...M.N][Mo.N+..M.NM...NM.NM]
```

```
[N-Mg.N+Mg.NM.NM.M..NM.NM.M][N-Mo.Mg..NM.NM...NM.NM]
```

```
[N-Mv.N-Mo.N.NM.M...NM.M][N+Mf.Mf.M.NM.NM..NM.NM.NM]
```

```
[N-Mg.N-Ml.M.NM.NM..M.NM.N][N+Mf.N-.NM.NM.N..NM.NM.NM]
```

```
[N-Mg.N-Mv.N.NM.M..N.NM.M][N-Mg.Ml..NM.NM...NM.NM]
```

```
[N-Mv.N-Mf.NM.NM.NM..NM.M.NM][Mf.N-Mn.N.M.NM..N.NM.NM]
```

```
[N-Mn.N+Mn.NM.M.NM..NM.M.N][Mv.N-.N.NM.N..N.NM.NM]
```

## Idea 4: constraints as relations

OT constraints are relations that insert violation marks.

```
xfst[17]: regex [C11 .o. SurSpell].l;
```

1.7 Kb. 19 states, 21 arcs, 4 paths.

```
xfst[18]: print words
```

```
[.mawa](.pa)
```

```
[.maw](a.pa)
```

```
[.ma](wa.pa)
```

```
[.m](awa.pa)
```

```
  _  _  _  _
```

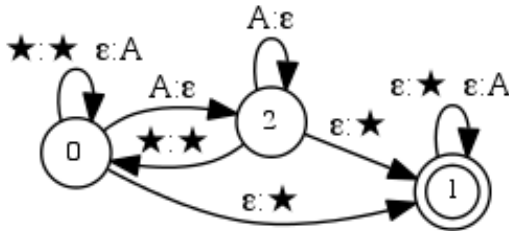
# SpHdLNaMinus

SpHdLNaMinus inserts a violation mark in a locus that heads a [N,-] span and does not start it. This is the last part of the four candidates, after marking.

1. . . NM . M] [N-Mo . Mo . . NM . NM . \* . . NM . M] [N-Mv . Mv . . NM . M . . . NM . NM]  
1. . . NM . M] [N-Mo . Mo . . NM . NM . \* . . NM . M] [N-Mv . Mv . . NM . M . . . NM . NM]  
1. . . NM . M] [N-Mo . Mo . . NM . NM . \* . . NM . M] [N-Mv . Mv . . NM . M . . . NM . NM]  
. NM . NM] [N-Mo . N-Mo . . NM . NM . . . NM . M] [N-Mv . Mv . . NM . M . . . NM . NM]

## Idea 5: Algebraic optimization

We want to optimize a set of candidates using a constraint. Optimization uses set difference and the relation  $LX(x, y)$  true iff  $x$  contains fewer violation marks than  $y$ .  $LX$  is definable in the finite state calculus. Here is the finite state machine that results from compilation. Pairs of strings for paths from state 0 to states 0 and 2 have equal numbers of stars. Paths from 0 to the final state 1 have more stars on the right.



# Algebraic optimization

We want to optimize a candidate set  $x$  with a constraint  $r$ .

$[x \circ r]'$	$x$ marked up using constraint $r$
$[x \circ r \circ LX]'$	Strings that have more marks than some candidate in $[x \circ r]'$ .
$[x \circ r] - [x \circ r \circ LX]$	Subtract off sub-optimal candidates to obtain the optimal candidates.

$x \circ r$  is the restriction of relation  $r$  to domain  $x$ ,  $r \circ r'$  is the composition of relations  $r$  and  $r'$ , and  $r'$  is the image of relation  $r$ .

Algebraic optimization was introduced in Eisner (2001). Using it with  $LX$  gives classical OT optimization. It only works for generation—this finite state realization of classical OT does not allow us to map surface representations to underlying ones.

## Example

```
xfst[24]: regex [C10 .o. SurSpell].l;  
3.0 Kb. 37 states, 51 arcs, 168 paths.  
xfst[25]: print random-words  
[.m][.a](wa.pa)  
[.m](a.w)(a.pa)  
[.m][.aw](a.p)(.a)  
[.m][.a](w.a)(.p)[.a]  
[.ma](.w)(a.p)(.a)  
[.m][.aw][.a](.pa)  
[.m][.a](wa.pa)
```

## Example

```
xfst[26]: regex [Optimize(C10,NoSpanNa) .o. SurSpell].l;  
1.7 Kb. 19 states, 21 arcs, 4 paths.  
xfst[27]: print words  
[.mawa](.pa)  
[.maw](a.pa)  
[.ma](wa.pa)  
[.m](awa.pa)
```



## Idea 6: Constraint families as functions

Constraint families are an organizing principle in OT phonology. Instead of an single constraint NoSpanNa, we have a family of no-span constraints NoSpan(F), where F is a feature attribute.

```
# A head of an underlying [F X] span is also the head of a surface [F X] span.  
# [H(F,X) - h(F,X)] is a set difference, describing loci that  
# are underlying heads of [F X], but not surface heads of [F,X].  
# See Headed Spans and Autometrical Spreading (5).  
define FthHdSp(F,X) [...] -> Vio || .#. [H(F,X) - h(F,X)] _ ;
```

## Constraint families as functions

```
# Penalize the locus at the start of a surface F span.  
# With standard optimization, this is equivalent to McCarthy's *A-Span(F),  
# because it systematically assigns one more violation mark.  
# See Headed Spans and Autometrical Spreading (4).  
define NoSpan(F) [..] -> Vio || .#. i(F) _ ;
```

# Constraint families as functions

```
# Span head location.  
# Penalize a surface [F,X] head that is not span-initial.  
# The head is the locus.  
# See Headed Spans and Autometrical Spreading (13a,c).  
define SpHdL(F,X)  [..] -> Vio || .#. [h(F,X) - [Locus & i(F)]] _ ;
```

## Program for an OT derivation

```
define C0 Gen(U(A) U(f));  
  
define C1 Optimize(C0,NoExceptionNa);  
define C2 Optimize(C1,NoExceptionMa);  
define C3 Optimize(C2,FthMaNa);  
define C4 Optimize(C3,FthMaOb);  
define C5 Optimize(C4,FthMaFr);  
define C6 Optimize(C5,FthMaLi);  
define C7 Optimize(C6,FthMaGl);  
define C8 Optimize(C7,FthMaVo);  
define C9 Optimize(C8,NoSpanMa);  
define C10 Optimize(C9,OrHdFr);  
define C11 Optimize(C10,NoSpanNa);
```



## Underlying unit segment

```
## Surface and underlying unit segments, with no exception for the feature.  
define SUnitSeg(F,X) [l1v(F,X) & le0(F) & h1(F)] [r1(F) & re0(F)];  
define UUnitSeg(F,X) [L1v(F,X) & H1(F)] R1(F);
```

An underlying unit segment for  $[F,X]$  starts  $F$  with value  $X$ , heads  $F$ , and ends  $F$ .

## Underlying spelling for nasal problem

```
# Map an underlying "letter" to a set of segments.  
# a unit span. The lower side unconstrained.
```

```
# X is the nasality, Y is the manner  
define USeg(X,Y) UUnitSeg(N,X) & UUnitSeg(M,Y);
```

```
define UC Z ) Test([p & Z],USeg(Minus,o),  
  Test([f & Z],USeg(Minus,f),  
    Test([l & Z],USeg(Minus,l),  
      Test([w & Z],USeg(Minus,g),  
        Test([a & Z],USeg(Minus,v),  
          Test([M & Z],USeg(Plus,n),  
            _____
```

To spell “f” underlyingly, specify a unit [N,-] span and a unit [M,f] span.

## Underlying spelling for nasal problem

```
# Map an underlying "letter" to a set of segments.  
# a unit span. The lower side unconstrained.
```

```
# X is the nasality, Y is the manner  
define USeg(X,Y) UUnitSeg(N,X) & UUnitSeg(M,Y);
```

```
define U( Z ) Test([p & Z],USeg(Minus,o),  
  Test([f & Z],USeg(Minus,f),  
    Test([l & Z],USeg(Minus,l),  
      Test([w & Z],USeg(Minus,g),  
        Test([a & Z],USeg(Minus,v),  
          Test([M & Z],USeg(Plus,n),  
            _____
```

A sequence of conditionals spells various underlying letters, e.g. U(f).

## McCarthy conditional

```
## For set-denoting X,Y, and Z
## Test(X,Y,Z) = if (X is non-empty) then Y else Z.
## A relational X is reduced to its lower side.
## True is the universal set.
## False is the empty set.
```

```
define Test(X,Y,Z) [Z - [X .o. [* .x. ?*]].l] |
                    [Y - ~[[X .o. [* .x. ?*]].l]];
```

Test is a McCarthy conditional in the finite state calculus.

Test(X,Y,Z) is Y if X is empty, otherwise Z.



## Truth values

```
## Suppose X is empty. Then Test(X,Y,Z) = Z.  
# 1. X is empty                               Assumption  
# 2. [X .o. U].1 is empty                     1.  
# 3. Z - [X .o. U].1 = Z                     2.  
# 4. ~[X .o. U].1 is the universal set       2.  
# 5. [Y - ~[[X .o. U].1]] is empty           4.  
# 6. [Z - [X .o. U].1] | [Y - ~[[X .o. U].1]] = Z
```

```
define True ?*;  
define False ~True;
```

Empty language is used as False, and universal language (or any non-empty language) is used as True.

## Gen

```
define Gen(X) Wfw & X;
```

```
define C0 Gen(U(A) U(f));
```

```
define C1 Optimize(C0,NoExceptionNa);
```

```
define C2 Optimize(C1,NoExceptionMa);
```

```
define C3 Optimize(C2,FthMaNa);
```

## Partial spreading

```
define C0 Gen(U(M) U(a) U(w) U(a) U(p) U(a));  
define C1 Optimize(C0,NoExceptionNa);  
define C2 Optimize(C1,NoExceptionMa);  
define C3 Optimize(C2,FthMaNa);  
define C4 Optimize(C3,FthMaOb);  
define C5 Optimize(C4,FthMaFr);  
define C6 Optimize(C5,FthMaLi);  
define C7 Optimize(C6,FthMaGl);  
define C8 Optimize(C7,FthMaVo);
```

# Partial spreading

In shell:

```
xfst -q -f mawapa.fst
```

```
xfst -q -f mawapa2.fst
```

# Summary

1. Labeled brackets as representation of spans
2. Underlying and surface representations in one string
3. Build representations a sequence of definitions in finite state calculus
4. Constraints as relations that insert violation marks
5. Algebraic optimization
6. Constraint families as functions
7. Finite state program calculates directly with with candidate sets and constraints.
8. Encoding of transparent segments using embedded exception spans.